2014 26th Euromicro Conference on Real-Time Systems

Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks

Jing Li[†], Jian-Jia Chen[§], Kunal Agrawal[†], Chenyang Lu[†], Chris Gill[†], Abusayeed Saifullah[†]
[†]Washington University in St. Louis, U.S.A.
[§]TU Dortmund University, Germany
li.jing@wustl.edu, [§]jian-jia.chen@cs.uni-dortmund.de, {kunal, lu, cdgill}@cse.wustl.edu, saifullah@wustl.edu

Abstract

This paper considers the scheduling of parallel real-time tasks with implicit deadlines. Each parallel task is characterized as a general directed acyclic graph (DAG). We analyze three different real-time scheduling strategies: two well known algorithms, namely global earliest-deadline-first and global rate-monotonic, and one new algorithm, namely federated scheduling. The federated scheduling algorithm proposed in this paper is a generalization of partitioned scheduling to parallel tasks. In this strategy, each high-utilization task (utilization ≥ 1) is assigned a set of dedicated cores and the remaining low-utilization tasks share the remaining cores. We prove capacity augmentation bounds for all three schedulers. In particular, we show that if on unit-speed cores, a task set has total utilization of at most m and the critical-path length of each task is smaller than its deadline, then federated scheduling can schedule that task set on m cores of speed 2; G-EDF can schedule it with speed $\frac{3+\sqrt{5}}{2} \approx 2.618$; and G-RM can schedule it with speed $2+\sqrt{3} \approx 3.732$. We also provide lower bounds on the speedup and show that the bounds are tight for federated scheduling and G-EDF when m is sufficiently large.

I. Introduction

In the last decade, multicore processors have become ubiquitous and there has been extensive work on how to exploit these parallel machines for real-time tasks. In the real-time systems community, there has been significant research on scheduling task sets with *inter-task parallelism* — where each task in the task set is a sequential program. In this case, increasing the number of cores allows us to increase the number of tasks in the task set. However, since each task can only use one core at a time, the computational requirement of a single task is still limited by the capacity of a single core. Recently, there has been some interest in the design and analysis of scheduling strategies for task sets with *intra-task parallelism* (in addition to inter-task parallelism), where individual tasks are parallel programs and can potentially utilize more than one core in parallel. These models enable tasks with higher execution demands and tighter deadlines, such as those used in autonomous vehicles [31], video surveillance, computer vision, radar tracking and real-time hybrid testing [28]

In this paper, we consider the general *directed acyclic graph (DAG)* model. We analyze three different scheduling strategies: a new strategy, namely federated scheduling, and two classic strategies, namely global EDF and global rate-monotonic scheduling. We prove that all three strategies provide strong performance guarantees, in the form of *capacity augmentation bounds*, for scheduling parallel DAG tasks with implicit deadlines.

One can generally derive two types of performance bounds for real-time schedulers. The traditional bound is called a ***resource augmentation bound*** (also called a *processor speed-up factor*). A scheduler $\mathcal{S}$ provides a resource augmentation bound of $b \geq 1$ if it can successfully schedule any task set τ on m cores of speed b as long as the ideal scheduler can schedule τ on m cores of speed 1. A resource augmentation bound provides a good notion of how close a scheduler is to the optimal schedule, but it has a drawback. Note that the *ideal scheduler* is only a hypothetical scheduler, meaning that it always finds a feasible schedule if one exists. Unfortunately, since we often cannot tell whether the ideal scheduler can schedule a given task set on unit-speed cores, a resource augmentation bound may not provide a schedulability test.

Another bound that is commonly used for sequential tasks is a ***utilization bound***. A scheduler $\mathcal{S}$ provides a utilization bound of b if it can successfully schedule any task set which has total utilization at most m/b on m cores.[1] A utilization bound provides more information than a resource augmentation bound; any scheduler that guarantees a utilization bound of b automatically guarantees a

[1]A utilization bound is often stated in terms of $1/b$; we adopt this notation in order to be consistent with the other bounds stated here.

1068-3070/14 $31.00 © 2014 IEEE
DOI 10.1109/ECRTS.2014.23

resource augmentation bound of b as well. In addition, it acts as a very simple schedulability test in itself, since the total utilization of the task set can be calculated in linear time and compared to m/b. Finally, a utilization bound gives an indication of how much load a system can handle; allowing us to estimate how much over-provisioning may be necessary when designing a platform. Unfortunately, it is often impossible to prove a utilization bound for parallel systems due to Dhall's effect; often, we can construct pathological task sets with utilization arbitrarily close to 1, but which cannot be scheduled on m cores.

Li et al. [35] defined a concept of ***capacity augmentation bound*** which is similar to the utilization bound, but adds a new condition. A scheduler $\mathcal{S}$ provides a capacity augmentation bound of b if it can schedule any task set τ which satisfies the following two conditions: (1) the total utilization of τ is at most m/b, and (2) the worst-case critical-path length of each task L_i (execution time of the task on an infinite number of cores)[2] is at most $1/b$ fraction of its deadline. A capacity augmentation bound is quite similar to a utilization bound: it also provides more information than a resource augmentation bound does; any scheduler that guarantees a capacity augmentation bound of b automatically guarantees a resource augmentation bound of b as well. It also acts as a very simple schedulability test. Finally, it can also provide an estimation of the load a system is expected to handle.

There has been some recent research on proving both resource augmentation bounds and capacity augmentation bounds for various scheduling strategies for parallel tasks. This work falls in two categories. In *decomposition-based strategies*, the parallel task is decomposed into a set of sequential tasks and they are scheduled using existing strategies for scheduling sequential tasks on multiprocessors. In general, decomposition-based strategies require explicit knowledge of the structure of the DAG off-line in order to apply decomposition. In non-decomposition based strategies, the program can unfold dynamically since no off-line knowledge is required.

For a decomposed strategy, most prior work considers *synchronous tasks* (subcategory of general DAGs) with implicit deadlines. Lakshmanan et al. [32] proved a capacity augmentation bound of 3.42 for partitioned fixed-priority scheduling for a restricted category of synchronous tasks [3] under decomposed deadline monotonic scheduling. Saifullah et al. [45] provide a different decomposition strategy for general parallel synchronous tasks and prove a capacity augmentation bound of 4 when the decomposed tasks are scheduled using global EDF and 5 when scheduled using partitioned DM. Kim et al. [31] provide another decomposition strategy and prove a capacity augmentation bound of 3.73 using global deadline monotonic scheduling. Nelissen et al. [40] proved a resource augmentation bound of 2 for general synchronous tasks. More recently, Saifullah et al. [44] provide a decomposition strategy for general DAG tasks that provides a capacity augmentation bound of 4.

For non-decomposition strategies, researchers have studied primarily global earliest deadline first (***G-EDF***) and global rate-monotonic (***G-RM***). Andersson and Niz [4] show that G-EDF provides resource augmentation bound of 2 for synchronous tasks with constrained deadlines. Both Li et al. [35] and Bonifaci et al. [15] concurrently showed that G-EDF provides a resource augmentation bound of 2 for general DAG tasks with arbitrary deadlines. In their paper, Bonifaci et al. also proved that G-RM provides a resource augmentation bound of 3 for parallel DAG tasks with arbitrary deadlines; Li et al. also provide a capacity augmentation bound of 4 for G-EDF for task sets with implicit deadlines.

In summary, the best known capacity augmentation bound for implicit deadlines tasks are 4 for DAG tasks using G-EDF, and 3.73 for parallel synchronous tasks using decomposition combined with G-DM. The contributions of this paper are as follows:

1 We propose a novel ***federated scheduling strategy***. Here, each *high-utilization task* (utilization ≥ 1) is allocated a dedicated cluster (set) of cores. A multiprocessor scheduling algorithm is used to schedule all *low-utilization tasks*, each of which is run sequentially, on a shared cluster composed of the remaining cores. Federated scheduling can be seen as a partitioned strategy generalized to parallel tasks. This is the best known capacity augmentation bound for *any scheduler* for parallel DAGs.

2 We prove that the capacity augmentation bound for this federated scheduler is 2. In addition, we also show no scheduler can provide a better capacity augmentation bound of $2 - 1/m$ for parallel tasks. Therefore, a bound of 2 for federated scheduling is tight when m is large enough.

3 We improve the capacity augmentation bound of G-EDF to $\frac{3+\sqrt{5}}{2} \approx 2.618$ for DAGs. When m is large, there is a matching lower bound for G-EDF [35]; hence, this result closes the gap for large m. This is the best known capacity augmentation bound for *any global scheduler* for parallel DAGs.

4 We show that G-RM has a capacity augmentation bound of $2 + \sqrt{3} \approx 3.732$. This is the best known capacity augmentation bound for *any fixed-priority scheduler* for DAG tasks. Even if restricted to synchronous tasks, this is still the best bound for global fixed priority scheduling without decomposition.

The paper is organized as follows. Section II defines the DAG model for parallel tasks and provides some definitions. Section III presents our federated scheduling

[2]Critical-path length of a sequential task is equal to its execution time

[3]Fork-join task model, in their terminology.

algorithm and proves the augmentation bound. Section IV proves a lower bound for any scheduler for parallel tasks. Section V presents a canonical form to give an upper bound of the work of a DAG that should be done in a specified interval length. Section VI proves that G-EDF provides a capacity augmentation bound of 2.618. Section VII shows that G-RM provides a capacity augmentation bound of 3.732. We discuss some practical considerations of the three schedulers in Section VIII. Section IX discusses related work and Section X concludes this paper.

II. System Model

We now present the details of the DAG task model for parallel tasks and some additional definitions.

We consider a set τ of n independent sporadic real-time tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. A task τ_i represents an infinite sequence of arrivals and executions of task instances (also called jobs). We consider the *sporadic task model* [9, 29] where, for a task τ_i, the *minimum inter-arrival time* (or *period*) T_i represents the time between consecutive arrivals of task instances, and the *relative deadline* D_i represents the temporal constraint for executing the job. If a task instance of τ_i arrives at time t, the execution of this instance must be finished no later than the *absolute deadline* $t+D_i$ and the release of the next instance of task τ_i must be no earlier than t plus the minimum inter-arrival time, i.e. $t+T_i$. In this paper, we consider *implicit deadline tasks* where each task τ_i's relative deadline D_i is equal to its minimum inter-arrival time T_i; that is, $T_i = D_i$. We consider the schedulability of this task set on a uniform multicore system consisting of m identical cores.

Each task $\tau_i \in \tau$ is a parallel task and is characterized as a *directed acyclic graph (DAG)*. Each node (subtask) in the DAG represents a sequence of instructions (a thread) and each edge represents a dependency between nodes. A node (subtask) is *ready* to be executed when all its predecessors have been executed. Throughout this paper, as it is not necessary to build the analysis based on specific structure of the DAG, only two parameters related to the execution pattern of task τ_i are defined:

- *total execution time (or work)* C_i of task τ_i: This is the summation of the worst-case execution times of all the subtasks of task τ_i.
- *critical-path length* L_i of task τ_i: This is the length of the critical-path in the given DAG, in which each node is characterized by the worst-case execution time of the corresponding subtask of task τ_i. Critical-path length is the worst-case execution time of the task on an infinite number of cores.

Given a DAG, obtaining work C_i and critical-path length L_i [46, pages 661-666] can both be done in linear time.

For brevity, the *utilization* $\frac{C_i}{T_i} = \frac{C_i}{D_i}$ of task τ_i is denoted by u_i for implicit deadlines. The total utilization of the task set is $U_\sum = \sum_{\tau_i \in \tau} u_i$.

Utilization-Based Schedulability Test. In this paper, we analyze schedulers in terms of their capacity augmentation bounds. The formal definition is presented here:

Definition 1. *Given a task set τ with total utilization of $U_\sum$, a scheduling algorithm $\mathcal{S}$ with* ***capacity augmentation bound*** *b can always schedule this task set on m cores of speed b as long as τ satisfies the following conditions on unit speed cores.*

$$\textit{Utilization does not exceed total cores, } \sum_{\tau_i \in \tau} u_i \leq m \quad (1)$$

$$\textit{For each task } \tau_i \in \tau, \textit{the critical path } L_i \leq D_i \quad (2)$$

Since no scheduler can schedule a task set τ on m unit speed cores unless Conditions (1) and (2) are met, a capacity augmentation bound automatically leads to a resource augmentation bound. This definition can be equivalently stated (without reference to the speedup factor) as follows: Condition (1) says that the total utilization $U_\sum$ is at most m/b and Condition (2) says that the critical-path length of each task is at most $1/b$ of its relative deadline, that is, $L_i \leq D_i/b$. Therefore, in order to check if a task set is schedulable we only need to know the total task set utilization, and the maximum critical-path utilization. Note that a scheduler with a smaller b is better than another with a larger b, since when $b = 1$ $\mathcal{S}$ is an optimal scheduler.

III. Federated Scheduling

This section presents the federated scheduling strategy for parallel tasks with implicit deadlines. We prove that it provides a capacity augmentation bound of 2 on m-core machine parallel real-time tasks.

A. Federated Scheduling Algorithm

Given a task set τ, the *federated scheduling algorithm* works as follows: First, tasks are divided into two disjoint sets: τ_{high} contains all ***high-utilization tasks*** — tasks with worst-case utilization at least one ($u_i \geq 1$), and τ_{low} contains all the remaining ***low-utilization tasks***. Consider a high-utilization task τ_i with worst-case execution time C_i, worst-case critical-path length L_i, and deadline D_i (which is equal to its period T_i). We assign n_i dedicated cores to τ_i, where n_i is

$$n_i = \left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil \quad (3)$$

We use $n_{\text{high}} = \sum_{\tau_i \in \tau_{\text{high}}} n_i$ to denote the total number of cores assigned to high-utilization tasks τ_{high}. We assign the remaining cores to all low-utilization tasks τ_{low}, denoted as $n_{\text{low}} = m - n_{\text{high}}$. The federated scheduling algorithm *admits* the task set τ, if n_{low} is non-negative and $n_{\text{low}} \geq 2\sum_{\tau_i \in \tau_{\text{low}}} u_i$.

After a valid core allocation, *runtime scheduling* proceeds as follows: (1) Any greedy (work-conserving) parallel scheduler can be used to schedule a high-utilization task

τ_i on its assigned n_i cores. Informally, a ***greedy scheduler*** is one that never keeps a core idle if some node is ready to execute. (2) Low-utilization tasks are treated and executed as though they are sequential tasks and any multiprocessor scheduling algorithm (such as partitioned EDF [37], or various rate-monotonic schedulers [3]) with a utilization bound of at most $1/2$ can be used to schedule all the low-utilization tasks on the allocated n_{low} cores. The important observation is that we can safely treat low-utilization tasks as sequential tasks since $C_i \leq D_i$ and parallel execution is not required to meet their deadlines.[4]

B. Capacity Augmentation Bound of 2 for Federated Scheduling

Theorem 1. *The federated scheduling algorithm has a capacity augmentation bound of* 2*.*

To prove Theorem 1, we consider a task set τ that satisfies Conditions (1) and (2) from Definition 1 for b =2. Then, we (1) state the relatively obvious Lemma 1; (2) prove that a high utilization task τ_i meets its deadline when assigned n_i cores; and (3) show that n_{low} is non-negative and satisfies $n_{\text{low}} \geq b\sum_{\tau_i \in \tau_{\text{low}}} u_i$ and therefore all low utilization tasks in τ will meet deadlines when scheduled using any multiprocessor scheduling strategy with utilization bound no less than b (i.e. can afford total task set utilization of $m/b = 50\%m$). These three steps complete the proof.

Lemma 1. *A task set τ is classified into disjoint subsets $s_1, s_2, ..., s_k$, and each subset is assigned a dedicated cluster of cores with size $n_1, n_2, ..., n_k$ respectively, such that $\sum_i n_i \leq m$. If each subset s_j is schedulable on its n_j cores using some scheduling algorithm $\mathcal{S}_j$ (possibly different for each subset), then the whole task set is guaranteed to be schedulable on m cores.*

High-Utilization Tasks Are Schedulable. Assume that a machine's execution time is divided into discrete quanta called *steps*. During each step a core can be either idle or performing one unit of work. We say a step is *complete* if no core is idle during that step, and otherwise we say it is *incomplete*. A *greedy* scheduler never keeps a cores idle if there is ready work available. Then, for a greedy scheduler on n_i cores, we can state two straightforward lemmas [35].

Lemma 2. [Li13] *Consider a greedy scheduler running on n_i cores for t time steps. If the total number of incomplete steps during this period is t^*, the total work F^t done during these time steps is at least $F^t \geq n_i t - (n_i - 1)t^*$.*

[4]Even if these tasks are expressed as parallel programs, it is easy to enforce correct sequential execution of parallel tasks — any topological ordered execution of the nodes of the DAG is a valid sequential execution.

Lemma 3. [Li13] *If a job of task τ_i is executed by a greedy scheduler, then every incomplete step reduces the remaining critical-path length of the job by 1.*

From Lemmas 2 and 3, we can establish Theorem 2.

Theorem 2. *If an implicit-deadline deterministic parallel task τ_i is assigned $n_i = \left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil$ dedicated cores, then all its jobs can meet their deadlines, when using a greedy scheduler.*

Proof: For contradiction, assume that some job of a high-utilization task τ_i misses its deadline when scheduled on n_i cores by a greedy scheduler. Therefore, during the D_i time steps between the release of this job and its deadline, there are fewer than L_i incomplete steps; otherwise, by Lemma 3, the job would have completed. Therefore, by Lemma 2, the scheduler must have finished at least $n_i D_i - (n_i - 1)L_i$ work.

$$\begin{aligned} & n_i D_i - (n_i - 1)L_i = n_i(D_i - L_i) + L_i \\ = \quad & \left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil (D_i - L_i) + L_i \\ \geq \quad & \frac{C_i - L_i}{D_i - L_i}(D_i - L_i) + L_i = C_i \end{aligned}$$

Since the job has work of at most C_i, it must have finished in D_i steps, leading to a contradiction. □

Low-Utilization Tasks are Schedulable. We first calculate a lower bound on n_{low}, the number of total cores assigned to low-utilization tasks, when a task set τ that satisfies Conditions (1) and (2) of Definition 1 for $b = 2$ is scheduled using a federated scheduling strategy.

Lemma 4. *The number of cores assigned to low-utilization tasks is at least $n_{low} \geq 2\sum_{low} u_i$.*

Proof: Here, for brevity of the proof, we denote $\sigma_i = \frac{D_i}{L_i}$. It is obvious that $D_i = \sigma_i L_i$ and hence $C_i = D_i u_i = \sigma_i u_i L_i$. Therefore,

$$n_i \quad = \quad \left\lceil \frac{C_i - L_i}{D_i - L_i} \right\rceil = \left\lceil \frac{\sigma_i u_i L_i - L_i}{\sigma_i L_i - L_i} \right\rceil = \left\lceil \frac{\sigma_i u_i - 1}{\sigma_i - 1} \right\rceil$$

Since each task τ_i in task set τ satisfies Condition (2) of Definition 1 for $b = 2$; therefore, the critical-path length of each task is at most $1/b$ of its relative deadline, that is, $L_i \leq D_i/b \implies \sigma_i \geq b = 2$.

By the definition of high-utilization task τ_i, we have $1 \leq u_i$. Together with $\sigma_i \geq 2$, we know that:

$$0 \leq \frac{(u_i - 1)(\sigma_i - 2)}{\sigma_i - 1}$$

From the definition of ceiling, we can derive

$$\begin{aligned} n_i &= \left\lceil \frac{\sigma_i u_i - 1}{\sigma_i - 1} \right\rceil \\ &< \frac{\sigma_i u_i - 1}{\sigma_i - 1} + 1 = \frac{\sigma_i u_i + \sigma_i - 2}{\sigma_i - 1} \\ &\leq \frac{\sigma_i u_i + \sigma_i - 2}{\sigma_i - 1} + \frac{(u_i - 1)(\sigma_i - 2)}{\sigma_i - 1} \\ &= \frac{\sigma_i u_i + \sigma_i - 2 + \sigma_i u_i - 2u_i - \sigma_i + 2}{\sigma_i - 1} \\ &= \frac{2\sigma_i u_i - 2u_i}{\sigma_i - 1} = \frac{2u_i(\sigma_i - 1)}{\sigma_i - 1} = 2u_i \\ &= bu_i \end{aligned}$$

In summary, for each high-utilization task, $n_i < bu_i$. So, their sum τ_{high} satisfies $n_{\text{high}} = \sum_{\text{high}} n_i < b \sum_{\text{high}} u_i$. Since the task set also satisfies Condition (1), we have

$$n_{\text{low}} = m - n_{\text{high}} > b \sum_{\text{all}} u_i - b \sum_{\text{high}} u_i = b \sum_{\text{low}} u_i$$

Thus, the number of remaining cores allocated to low-utilization tasks is at least $n_{\text{low}} > 2 \sum_{\text{low}} u_i$. □

Corollary 1. *For task sets satisfying Conditions (1) and (2), a multiprocessor scheduler with utilization bound of at least 50% can schedule all the low-utilization tasks sequentially on the remaining n_{low} cores.*

Proof: Low-utilization tasks are allocated n_{low} cores, and from Lemma 4 we know that the total utilization of the low utilization tasks is less than $n_{\text{low}}/b = 50\% n_{\text{low}}$. Therefore, any multiprocessor scheduling algorithm that provides a utilization bound of 2 (i.e., can schedule any task set with total worst-case utilization ratio no more than 50%) can schedule it. □

Many multiprocessor scheduling algorithms (such as partitioned EDF or partitioned RM) provide a utilization bound of $1/2$ (i.e., 50%) to sequential tasks. That is, given n_{low} cores, they can schedule any task set with a total worst-case utilization up to $n_{\text{low}}/2$. Using any of these algorithms for low-utilization tasks will guarantee that the federated algorithm meets all deadlines with capacity augmentation of 2.

Therefore, since we can successfully schedule both high and low-utilization tasks that satisfy Conditions (1) and (2), we have proven Theorem 1 (using Lemma 1).

As mentioned before, a capacity augmentation bound acts as a simple schedulability test. However, for federated scheduling, this test can be pessimistic, especially for tasks with high parallelism. Note, however, that the federated scheduling algorithm described in Section III-A can also be directly used as a (polynomial-time) schedulability test: given a task set, after assigning cores to each high-utilization task using this algorithm, if the remaining cores are sufficient for all low-utilization tasks, then the task set is schedulable and we can admit it without deadline misses. This schedulability test admits a strict superset of tasks admitted by the capacity augmentation bound test, and in practice, it may admits task sets with utilization greater than $m/2$.

IV. Lower Bound on Capacity Augmentation of Any Scheduler for Parallel Tasks

On a system with m cores, consider a task set τ with a single task, τ_1, which starts with sequential execution for $1-\epsilon$ time and then *forks* $\frac{m-1}{\epsilon}+1$ subtasks with execution time ϵ. Here, we assume ϵ is an arbitrarily small positive number. Therefore, the total work of task τ_1 is $C_1 = m$ and its critical-path length $L_i = 1$. The deadline (and also minimum inter-arrival time) of τ_1 is 1.

Theorem 3. *The capacity augmentation bound for any scheduler for parallel tasks on m cores is at least $2 - \frac{1}{m}$, when $\epsilon \to^+ 0$.*

Proof: Consider the system defined above. The finishing time of τ_1 by running at speed α is not earlier than $\frac{1-\epsilon}{\alpha} + \frac{\frac{m-1}{\epsilon}\epsilon}{m\alpha} = \frac{2}{\alpha} - \frac{1}{m\alpha} - \frac{\epsilon}{\alpha}$. If $\alpha > 2 - \frac{1}{m}$ and $\epsilon \to^+ 0$, then $\frac{2}{\alpha} - \frac{1}{m\alpha} - \frac{\epsilon}{\alpha} > 1$, then task τ_1 misses its deadline. Therefore, we reach the conclusion. □

Since Lemma 3 works for any scheduler for parallel tasks, we can conclude that the lower bound on capacity augmentation of federated scheduling is at least 2 , when m is sufficiently large. Since we have shown that the upper bound on capacity augmentation of federated scheduling is also 2, therefore, we have closed the gap between the lower and upper bound of federated scheduling for large m. Moreover, for sufficiently large m, federated scheduling has the best capacity augmentation bound, among all schedulers for parallel tasks.

V. Canonical Form of a DAG Task

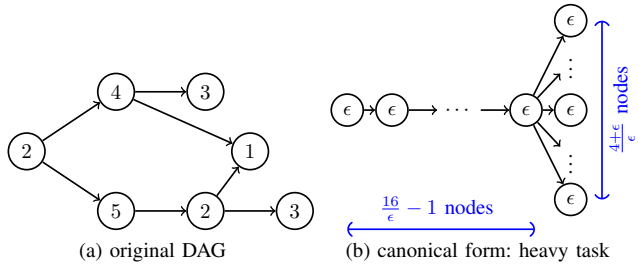
In this section, we introduce the concept of a DAG's *canonical form*. Note each task can have an arbitrarily complex DAG structure which may be difficult to analyze and may not even be known before runtime. However, given the known task set parameters (work, critical path length, utilization, critical-path utilization, etc.) we represent each task using a canonical DAG that allows us to upper bound the demand of the task in any given interval length t. These results will play an important role when we analyze the capacity augmentation bounds for G-EDF in Section VI and G-RM in Section VII. Recall that in this paper, we analyze tasks with implicit deadline, so period equals to deadline ($T_i = D_i$).

Recall that we classify each task τ_i as a *low-utilization* if $u_i = C_i/D_i < 1$ (and hence $C_i < D_i$); or *high-utilization* task, if τ_i's utilization $u_i \geq 1$.

For analytical purposes, instead of considering the complex DAG structure of individual tasks τ_i, we consider a *canonical form* τ_i^* of task τ_i. The canonical form of a task is represented by a simpler DAG. In particular, each subtask (node) of task τ_i^* has execution time ϵ, which is positive and arbitrarily small. Note that ϵ is a hypothetical unit-node execution time. Therefore, it is safe to assume that $\frac{D_i}{\epsilon}$ and $\frac{C_i}{\epsilon}$ are both integers. Low and high-utilization tasks have different canonical forms described below.

- The canonical form τ_i^* of a low-utilization task τ_i is simply a chain of C_i/ϵ nodes, each with execution time ϵ. Note that task τ_i^* is a sequential task.
- The canonical form τ_i^* of a high-utilization task τ_i starts with a chain of $D_i/\epsilon - 1$ nodes each with execution time ϵ. The total work of this chain is $D_i - \epsilon$. The last node of the chain *forks* all the remaining nodes. Hence, all the remaining $(C_i - D_i + \epsilon)/\epsilon$ nodes have an edge from the last node of this chain. Therefore, all these nodes can execute entirely in parallel.

Figure 1 provides an example for such a transformation for a high-utilization task. It is important to note that the canonical form τ_i^* does not depend on the DAG structure of τ_i at all. It depends only on the task parameters of τ_i.



(a) original DAG (b) canonical form: heavy task

Fig. 1: A high-utilization DAG task τ_i with $L_i = 12$, $C_i = 20$, $T_i = D_i = 16$, and $u_i = 1.25$ and its canonical form, where the number in each node is its execution time.

As an additional analysis tool, we define a hypothetical scheduling strategy $\mathcal{S}_\infty$ that must schedule a task set τ on an infinite number of cores, that is, $m = \infty$. With infinite number of cores, the prioritization of the sub-jobs becomes unnecessary and $\mathcal{S}$ can obtain an optimal schedule by simply assigning a sub-job to a core as soon as that sub-job becomes ready for execution. Using this schedule, all the tasks finish within their critical-path length; therefore, if $L_i \leq D_i$ for all tasks τ_i in τ, the task set always meets the deadlines. We denote this schedule as S_∞. Similarly, $S_{\infty,\alpha}$ is the resulting schedule when $\mathcal{A}_\infty$ schedules tasks on cores of speed $\alpha \geq 1$. Note that $S_{\infty,\alpha}$ finishes a job of task τ_i exactly L_i/α time units after it is released.

We now define some notations based on $S_{\infty,\alpha}$. Let $q_i(t,\alpha)$ be the total work finished by $S_{\infty,\alpha}$ between the arrival time r_i of task τ_i and time $r_i + t$. Therefore, in the interval from $r_i + t$ to $r_i + D_i$ (interval of length $D_i - t$) the remaining $C_i - q_i(t,\alpha)$ workload has to be finished.

We define *maximum load*, denoted by $work_i(t,\alpha)$, for task i as the maximum amount of work (computation) that $S_{\infty,\alpha}$ must do on the sub-jobs of τ_i in any interval of length t. We can derive $work_i(t,\alpha)$ as follows:

$$work_i(t,\alpha) = \begin{cases} C_i - q_i(D_i - t, \alpha) & t \leq D_i \\ \left\lfloor \frac{t}{D_i} \right\rfloor C_i + work_i(t - \left\lfloor \frac{t}{D_i} \right\rfloor D_i, \alpha) & t > D_i. \end{cases} \quad (4)$$

Clearly, both $q_i(t,\alpha)$ and $work_i(t,\alpha)$ for a task depend on the structure of the DAG.

We similarly define $q_i^*(t,\alpha)$ for the canonical form τ_i^*. As the canonical form in task τ_i^* is well-defined, we can derive $q_i^*(t,\alpha)$ directly. Note that ϵ can be arbitrarily small, and, hence, its impact is ignored when calculating $q_i^*(t,\alpha)$.

We can now define the canonical maximum load $work_i^*(t,\alpha)$ as the maximum workload of the canonical task τ_i^* in any interval t in schedule $S_{\infty,\alpha}$. For a low-utilization task τ_i, where $C_i/D_i < 1$, and τ_i^* is a chain, it is easy to see that the canonical workload is

$$work_i^*(t,\alpha) = \quad (5)$$
$$\begin{cases} 0 & t < D_i - \frac{C_i}{\alpha} \\ \alpha \cdot (t - (D_i - \frac{C_i}{\alpha})) & D_i - \frac{C_i}{\alpha} \leq t \leq D_i \\ \left\lfloor \frac{t}{D_i} \right\rfloor C_i + work_i^*(t - \left\lfloor \frac{t}{D_i} \right\rfloor D_i, \alpha) & t > D_i. \end{cases}$$

Similarly, for high-utilization tasks, where $C_i/D_i \geq 1$, when ϵ is arbitrarily small, we have

$$work_i^*(t,\alpha) = \quad (6)$$
$$\begin{cases} 0 & t < D_i - \frac{D_i}{\alpha} \\ C_i - D_i + \alpha \cdot (t - (D_i - \frac{D_i}{\alpha})) & D_i - \frac{D_i}{\alpha} \leq t \leq D_i \\ \left\lfloor \frac{t}{D_i} \right\rfloor C_i + work_i^*(t - \left\lfloor \frac{t}{D_i} \right\rfloor D_i, \alpha) & t > D_i. \end{cases}$$

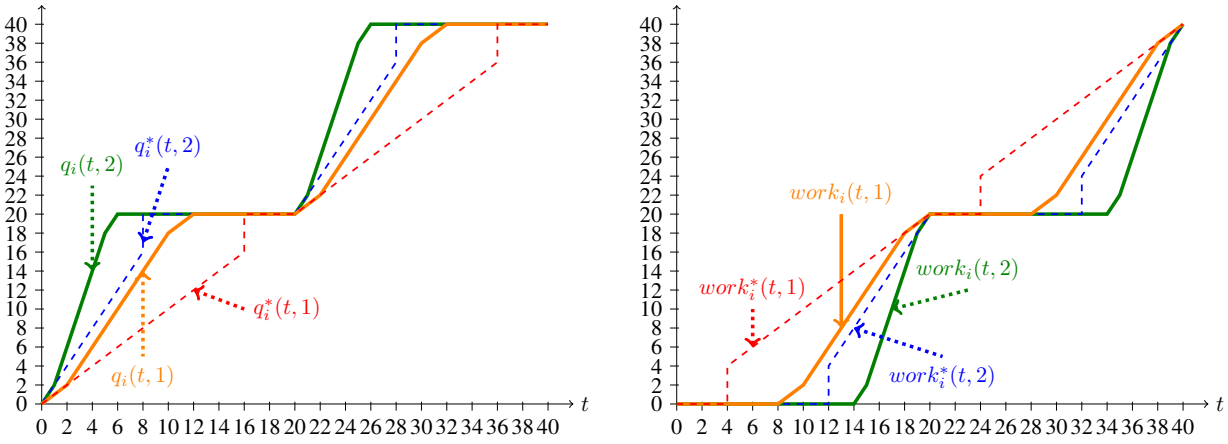
Figure 2 shows the $q_i^*(t,\alpha)$, $q_i(t,\alpha)$, $work_i^*(t,\alpha)$, and $work_i(t,\alpha)$ of the high-utilization task τ_i in Figure 1 when $D_i = 16$, $\alpha = 1$, and $\alpha = 2$. Note that $work_i^*(t,\alpha) \geq work_i(t,\alpha)$. In fact, the following lemma proves that $work_i^*(t,\alpha) \geq work_i(t,\alpha)$ for any $t > 0$ and $\alpha \geq 1$.

Lemma 5. *For any $t > 0$ and $\alpha \geq 1$, $work_i^*(t,\alpha) \geq work_i(t,\alpha)$.*

Proof: For low-utilization tasks, the entire work C_i is sequential. When $t < \frac{C_i}{\alpha}$, $q_i^*(t,\alpha)$ is αt, so $q_i(t,\alpha) \geq \alpha t = q_i^*(t,\alpha)$. When $\frac{C_i}{\alpha} \leq t < D_i$, $q_i(t,\alpha) = C_i = q_i^*(t,\alpha)$.

Similarly, for high-utilization tasks, the first D_i units of work is sequential, so when $t < \frac{D_i}{\alpha}$, $q_i^*(t,\alpha) = \alpha t$. In addition, $S_{\infty,\alpha}$ finishes τ_i exactly $\frac{L_i}{\alpha}$ time units after it is released, while it finishes the τ_i^* at $\frac{D_i}{\alpha}$. Since the critical-path length $L_i \leq D_i$ for all τ_i and τ_i^* at unit-speed system, when $t < \frac{L_i}{\alpha}$, $q_i(t,\alpha) \geq \alpha t = q_i^*(t,\alpha)$. When $\frac{L_i}{\alpha} \leq t < \frac{D_i}{\alpha}$, $q_i(t,\alpha) = q_i^*(\frac{D_i}{\alpha},\alpha) > q_i^*(t,\alpha)$, When $L_i < t \leq D_i$, Lastly, when $\frac{D_i}{\alpha} \leq t < D_i$, $q_i(t,\alpha) = C_i = q_i^*(t,\alpha)$

We can conclude that $q_i^*(t) \leq q_i(t)$ for any $0 \leq$



(a) $q_i^*(t,\alpha)$ and $q_i(t,\alpha)$ (b) $work_i^*(t,\alpha)$ and $work_i(t,\alpha)$

Fig. 2: $q_i^*(t,\alpha)$, $q_i(t,\alpha)$, $work_i^*(t,\alpha)$ and $work_i(t,\alpha)$ for the high-utilization task τ_i with $D_i = 20$ in Figure 1.

$t < D_i$. Combining with the definition of $work(t,\alpha)$ (Equation (4)), we complete the proof. □

We classify task τ_i as a *light* or *heavy* task. A task is a light task if $u_i = C_i/D_i < \alpha$. Otherwise, we say that τ_i is heavy ($u_i \geq \alpha$) . The following lemmas provide an upper bound on the density (the ratio of workload that has to be finished to the interval length) for heavy and light tasks.

Lemma 6. *For any task τ_i, $t > 0$ and $1 < \alpha$, we have*

$$\frac{work_i(t,\alpha)}{t} \leq \frac{work_i^*(t,\alpha)}{t} \leq \begin{cases} u_i & (0 \leq u_i < \alpha) \\ \frac{u_i - 1}{1 - \frac{1}{\alpha}} & (\alpha \leq u_i) \end{cases} \quad (7)$$

Proof: The first inequality in Inequality (7) comes from Lemma 5. We now show that the second inequality also holds for any task. Note that the right hand side is positive, since $1 > u_i > 0$. There are two cases:

Case 1: $0 < t \leq D_i$.

- If τ_i is a low-utilization task, where $work_i^*(t,\alpha)$ is defined in Equation (5). For any $0 < t \leq D_i$, we have

$$work_i^*(t,\alpha) - \frac{C_i}{D_i} \cdot t \leq \alpha(t - D_i + \frac{C_i}{\alpha}) - \frac{C_i}{D_i} \cdot t$$
$$= (t - D_i)(\alpha - \frac{C_i}{D_i}) \leq 0$$

  where we rely on assumptions: (a) $t \leq D_i$; (b) since τ_i is a low-utilization task, $\frac{C_i}{D_i} < 1$; and (c) $\alpha > 1$. Then, $\frac{work_i^*(t,\alpha)}{t} \leq \frac{C_i}{D_i} = u_i$.
- If τ_i is a high-utilization task, where $work_i^*(t,\alpha)$ is defined in Equation (6). Inequality (7) holds trivially when $0 < t < D_i - \frac{D_i}{\alpha}$, since the left side is 0 and right side is positive. For $D_i - \frac{D_i}{\alpha} \leq t \leq D_i$, we have

$$\frac{work_i^*(t,\alpha)}{t} = \frac{C_i + \alpha t - \alpha D_i}{t} = \alpha + D_i(\frac{u_i - \alpha}{t}),$$

  Therefore, $\frac{work_i^*(t,\alpha)}{t}$ is maximized either (a) when $t = D_i - \frac{D_i}{\alpha}$ if $u_i - \alpha \geq 0$ or (b) $t = D_i$ if $u_i - \alpha < 0$.

  If τ_i is a light task with $1 \leq u_i < \alpha$ and hence (b) is true, then we have $\frac{work_i^*(D_i,\alpha)}{D_i} = u_i$.

  If heavy task τ_i with $\alpha \leq u_i$ and hence (a) is true, then

$$\frac{work_i^*(D_i - \frac{D_i}{\alpha}, \alpha)}{D_i - \frac{D_i}{\alpha}} = \frac{C_i - D_i}{D_i - \frac{D_i}{\alpha}} = \frac{u_i - 1}{1 - \frac{1}{\alpha}}$$

  Therefore, Inequality (7) holds for $0 < t \leq D_i$.

Case 2: $t > D_i$ — Suppose that t is $kD_i + t'$, where k is $\left\lfloor \frac{t}{D_i} \right\rfloor$ and $0 < t' \leq D_i$. When $u_i < \alpha$, by Equation (5) and Equation (6), we have

$$\frac{work_i^*(t,\alpha)}{t} = \frac{kC_i + work_i^*(t',\alpha)}{kD_i + t'} \leq \frac{ku_iD_i + u_it'}{kD_i + t'} = u_i$$

When $\alpha \leq u_i$, we can derive that $u_i \leq \frac{u_i - 1}{1 - \frac{1}{\alpha}}$. By Equation (6), we have

$$\frac{work_i^*(t,\alpha)}{t} = \frac{kC_i + work_i^*(t',\alpha)}{kD_i + t'} \leq \frac{ku_iD_i + \frac{u_i - 1}{1 - \frac{1}{\alpha}}t'}{kD_i + t'}$$
$$\leq \frac{k\frac{u_i - 1}{1 - \frac{1}{\alpha}}D_i + \frac{u_i - 1}{1 - \frac{1}{\alpha}}t'}{kD_i + t'} \leq \frac{u_i - 1}{1 - \frac{1}{\alpha}}$$

Hence, Inequality (7) holds for any task and any $t > 0$. □

We denote τ_{L} and τ_{H} as the set of light and heavy tasks in a task set, respectively; $\|\tau_{\text{H}}\|$ as the number of heavy tasks in the task set; and total utilization of light and heavy tasks as $U_{\text{L}} = \sum_{\tau_{\text{L}}} u_i$ and $U_{\text{H}} = \sum_{\tau_{\text{H}}} u_i$, respectively.

Lemma 7. *For any task set, the following inequality holds:*

$$W = (\sum_{i=1}^{n} work_i(t,\alpha)) \leq \left(\frac{\alpha \cdot U_\Sigma - U_L - \alpha \cdot \|\tau_H\|}{\alpha - 1}\right) \cdot t \quad (8)$$
$$\leq \left(\frac{\alpha m - \alpha}{\alpha - 1}\right) \cdot t \quad (9)$$

Proof: By Lemma 6, for any $\alpha > 1$, it is clear that

$$\frac{W}{t} = \sum_{\tau_{\text{L}} + \tau_{\text{H}}} \sup_{t>0} \frac{work_i^*(t,\alpha)}{t} \leq \sum_{\tau_{\text{L}}} u_i + \frac{\sum_{\tau_{\text{H}}}(u_i - 1)}{1 - \frac{1}{\alpha}}$$
$$= \frac{\alpha \cdot \sum_{\tau_{\text{L}}} u_i - \sum_{\tau_{\text{L}}} u_i + \alpha \cdot \sum_{\tau_{\text{H}}} u_i - \alpha \cdot \sum_{\tau_{\text{H}}} 1}{\alpha - 1} = \frac{\alpha \cdot U_\Sigma - U_{\text{L}} - \alpha \cdot \|\tau_{\text{H}}\|}{\alpha - 1}$$

where sup is the supremum of a set of numbers, τ_{L} and τ_{H} are the sets of heavy tasks ($u_i \geq \alpha$) and light tasks ($u_i < \alpha$), respectively.

Note that $\sum_{\tau_{\mathrm{L}}} u_i + \sum_{\tau_{\mathrm{H}}} u_i = U_{\mathrm{L}} + U_{\mathrm{H}} = U_{\sum} \leq m$. Since for $\tau_i \in \tau_{\mathrm{H}}$, $u_i \geq \alpha$, $U_{\mathrm{H}} = \sum_{\tau_{\mathrm{H}}} u_i \geq \|\tau_{\mathrm{H}}\|\alpha$, we can derive the following upper bound:

$$U_{\sum} - U_{\mathrm{L}} - \|\tau_{\mathrm{H}}\|\alpha \leq \sup_{\tau}(U_{\sum} - U_{\mathrm{L}} - \|\tau_{\mathrm{H}}\|\alpha) = U_{\sum} - \alpha$$

This is because for any task set, there are two cases:

- If $\|\tau_{\mathrm{H}}\| = 0$ and hence $U_{\sum} = U_{\mathrm{L}}$, then $U_{\sum} - U_{\mathrm{L}} - \|\tau_{\mathrm{H}}\|\alpha = 0$.
- If $\|\tau_{\mathrm{H}}\| \geq 1$, then $U_{\sum} - U_{\mathrm{L}} = U_{\mathrm{H}} \leq U_{\sum}$ and $\|\tau_{\mathrm{H}}\|\alpha \geq \alpha$. Therefore, $U_{\sum} - U_{\mathrm{L}} - \|\tau_{\mathrm{H}}\|\alpha \leq U_{\sum} - \alpha$

Together with the definition of $U_{\sum}$ and $\sum_{\tau_{\mathrm{H}}} 1 = \|\tau_{\mathrm{H}}\|$,

$$W \leq \left(\frac{\alpha \cdot U_{\sum} - U_{\mathrm{L}} - \alpha \cdot \|\tau_{\mathrm{H}}\|}{\alpha - 1}\right) t \leq \left(\frac{\alpha \cdot U_{\sum} - \alpha}{\alpha - 1}\right) t \leq \left(\frac{\alpha m - \alpha}{\alpha - 1}\right) t$$

which proves the Inequalities 8 and 9 of Lemma 7. □

We use Lemma 7 in Sections VI and VII to derive bounds on G-EDF and G-RM scheduling.

VI. Capacity Augmentation of Global EDF

In this section, we use the results from Section V to prove the capacity augmentation bound of $(3+\sqrt{5})/2$ for G-EDF scheduling of parallel DAG tasks. In addition, we also show a matching lower bound when $m \geq 3$.

A. Upper Bound on Capacity Augmentation of G-EDF

Our analysis builds on the analysis used to prove the resource augmentation bounds by Bonifaci et al. [15]. We first review the particular lemma from the paper that we will use to achieve our bound.

Lemma 8. *If $\forall t > 0$, $(\alpha m - m + 1) \cdot t \geq \sum_{i=1}^{n} work_i(t, \alpha)$, the task set is schedulable by G-EDF on speed-α cores.*

Proof: This is based on a reformulation of Lemma 3 and Definition 10 in [15] considering cores with speed α. □

Theorem 4. *The capacity augmentation bound for G-EDF is $\frac{3-\frac{2}{m}+\sqrt{5-\frac{8}{m}+\frac{4}{m^2}}}{2}$ ($\approx \frac{3+\sqrt{5}}{2}$, when m is large).*

Proof: From Lemma 7 Inequality (9), we have $\forall t > 0$, $\sum_{i=1}^{n} work_i(t, \alpha) \leq (\frac{\alpha m - \alpha}{\alpha - 1}) \cdot t$. If $\frac{\alpha \cdot m - \alpha}{\alpha - 1} \leq (\alpha m - m + 1)$, by Lemma 8 the schedulability test for G-EDF holds. To calculate α, we solve the derived equivalent inequality

$$m\alpha^2 - (3m - 2)\alpha + (m - 1) \geq 0.$$

which solves to $\alpha = \left(3 - \frac{2}{m} + \sqrt{5 - \frac{8}{m} + \frac{4}{m^2}}\right)/2$. □

We now state a more general corollary relating the more precise bound using more information of a task set.

Corollary 2. *If a task set has total utilization $U_{\sum}$, the total heavy task utilization U_H and the number of heavy task $\|\tau_H\|$, then this task set will be schedulable under G-EDF on a m-core machine with speed $\alpha = \frac{2+\frac{U_{\sum}-\|\tau_H\|-1}{m}+\sqrt{\frac{4(U_H-\|\tau_H\|)}{m}+\frac{(U_{\sum}-\|\tau_H\|-1)^2}{m^2}}}{2}$.*

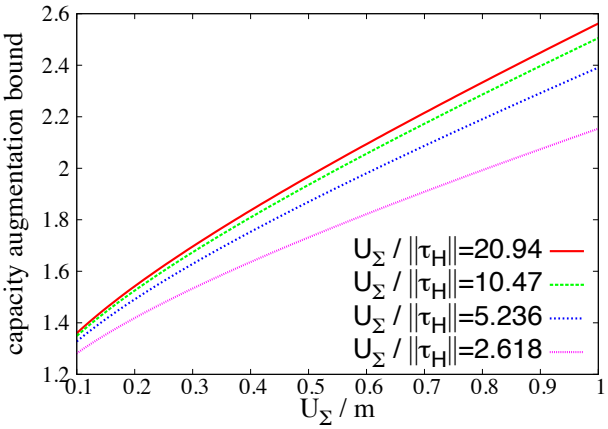


Fig. 3: The required speedup of G-EDF when m is sufficiently large and $U_{\mathrm{L}} = 0$ (i.e. $U_{\sum} = U_{\mathrm{H}}$).

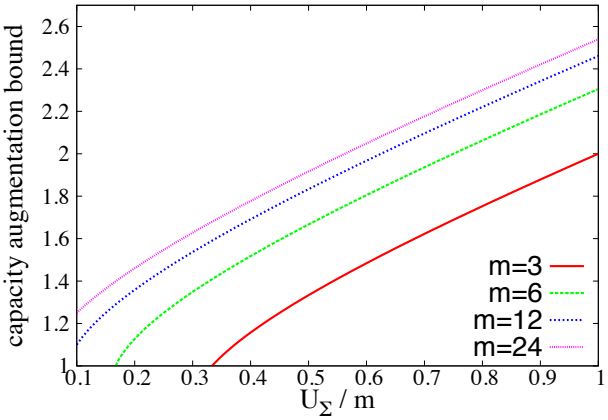


Fig. 4: The required speedup of G-EDF when $\|\tau_{\mathrm{H}}\| = 1$ and $U_{\mathrm{L}} = 0$ (i.e. $U_{\sum} = U_{\mathrm{H}}$).

Proof: The proof is the same as in the proof of Theorem 4, but without using the Inequality (9). Instead, we directly use Inequality (8). If $\frac{\alpha \cdot U_{\sum} - U_{\mathrm{L}} - \alpha \cdot \|\tau_{\mathrm{H}}\|}{\alpha - 1} \leq (\alpha m - m + 1)$, by Lemma 8 the schedulability test for G-EDF holds for this task set. Solving this, we can get the required speedup α for the schedulability of the task set.

However, note that heavy tasks are defined as the set of all tasks τ_i with utilization $u_i \geq \alpha$. Therefore, given a task set, to accurately calculate α, we start with the upper bound on α, which is $\hat{\alpha} = \left(3 - \frac{2}{m} + \sqrt{5 - \frac{8}{m} + \frac{4}{m^2}}\right)/2$; then for each iteration i, we can calculate the required speedup α^i by using the U_{H}^{i-1} and $\|\tau_{\mathrm{H}}\|^{i-1}$ from the $(i-1)$-th iteration; we iteratively classify more tasks into the set of heavy tasks and we stop when no more tasks can be added to this set, i.e., $\|\tau_{\mathrm{H}}\|^{i-1} = \|\tau_{\mathrm{H}}\|^{i}$. Through these iterative steps, we can calculate an accurate speedup. □

Figure 3 illustrates the required speedup of G-EDF provided in Corollary 2 when m is sufficiently large (i.e., $m = \infty$ and $1/m = 0$) and $U_{\mathrm{L}} = 0$ (i.e. $U_{\sum} = U_{\mathrm{H}}$). We vary $\frac{U_{\sum}}{m}$ and $\frac{U_{\sum}}{\|\tau_{\mathrm{H}}\|}$. Note that $\frac{U_{\sum}}{\|\tau_{\mathrm{H}}\|} = \frac{U_{\mathrm{H}}}{\|\tau_{\mathrm{H}}\|}$ is the average utilization of all heavy tasks, which should be no less than $(3+\sqrt{5})/2 \approx 2.618$. It can be also be seen that the bound is getting closer to $(3+\sqrt{5})/2$, when $\frac{U_{\sum}}{\|\tau_{\mathrm{H}}\|}$ is larger, which results from $\|\tau_{\mathrm{H}}\| = 1$ and $m = \infty$.

Figure 4 illustrates the required speedup of G-EDF provided in Corollary 2 when $\|\tau_{\mathrm{H}}\| \leq 1$ and $U_{\mathrm{L}} = 0$ (i.e. $U_{\sum} = U_{\mathrm{H}}$) with varying m. Note that $\alpha > 1$ is required by the proof of Corollary 2. And $\|\tau_{\mathrm{H}}\| = 1$ can only be true, if $\frac{U_{\sum}}{m} \geq \frac{\alpha}{m}$ (i.e. $\frac{U_{\sum}}{m} \geq 1/3$ for $m = 3$).

B. Lower Bound on Capacity Augmentation of G-EDF

As mentioned above, Li et al.'s lower bound [35] demonstrates the tightness of the above bound for large m. We now provide a lower bound for the capacity augmentation bound for small m.

Consider a task set τ with two tasks, τ_1 and τ_2. Task τ_1 starts with sequential execution for $1-\epsilon$ time and then *forks* $\frac{m-2}{\epsilon}+1$ subtasks with execution time ϵ. Here, we assume ϵ is an arbitrarily small positive number and hence it is safe to assume that $\frac{1}{\epsilon m}$ is a positive integer. Therefore, the total work of task τ_1 is $C_1 = m-1$ and its critical-path length is $L_i = 1$. The minimum inter-arrival time of τ_1 is 1.

Task τ_2 is simply a sequential task with work (execution time) of $1-\frac{1}{\alpha}$ and minimum inter-arrival time also $1-\frac{1}{\alpha}$, where $\alpha > 1$ will be defined later. Clearly, the total utilization is m and the critical-path length of each task is at most the relative deadline (minimum inter-arrival time).

Lemma 9. *When* $\alpha < \frac{3-\frac{2}{m}-\delta+\sqrt{5-\frac{12}{m}+\frac{4}{m^2}}}{2}$ *and* $\delta = 2\epsilon + g(\epsilon)$ *and* $m \geq 3$*, then* $\frac{1-2\epsilon}{\alpha} + \frac{m-2}{m\alpha} > 1 - \frac{1-\frac{1}{\alpha}}{\alpha}$ *holds.*

Proof: By solving $\frac{1-2\epsilon}{\alpha} + \frac{m-2}{m\alpha} = 1 - \frac{1-\frac{1}{\alpha}}{\alpha}$, we know that the equality holds when

$$\alpha < \frac{3-\frac{2}{m}-2\epsilon+\sqrt{5-\frac{12}{m}+\frac{4}{m^2}}-g(\epsilon)}{2}$$

where $g(\epsilon)$ is a positive function of $\frac{1}{\epsilon}$, which approaches to 0 when ϵ approaches 0. Now, by setting δ to $2\epsilon + g(\epsilon)$, we reach the conclusion. □

Theorem 5. *The capacity augmentation bound for G-EDF is at least* $\frac{3-\frac{2}{m}+\sqrt{5-\frac{12}{m}+\frac{4}{m^2}}}{2}$*, when* $\epsilon \to^+ 0$.

Proof: Consider the system with two tasks τ_1 and τ_2 defined in the beginning of Section VI-B. Suppose that the arrival of task τ_1 is at time 0, and the arrival of task τ_2 is at time $\frac{1}{\alpha} + \frac{\epsilon}{\alpha}$. By definition, the first jobs of τ_1 and τ_2 have absolute deadlines at 1 and $1 + \frac{\epsilon}{\alpha}$. Hence, G-EDF will execute the sequential execution of task τ_1 and the sub-jobs of task τ_1 first, and then execute τ_2.

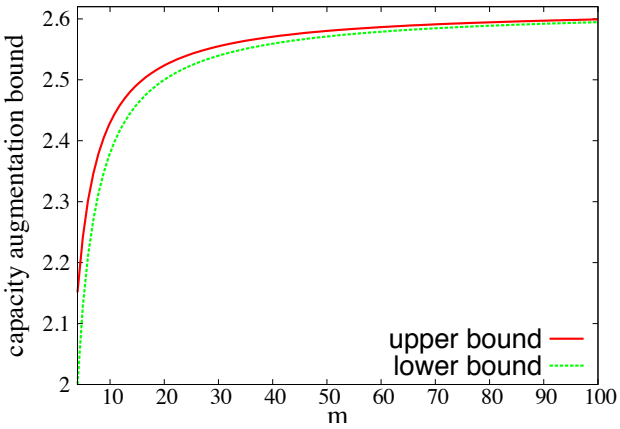


Fig. 5: The upper bound of G-EDF provided in Theorem 4 and the lower bound in Theorem 5 with respect to the capacity augmentation bound.

The finishing time of τ_1 at speed α is not earlier than $\frac{1-\epsilon}{\alpha} + \frac{\frac{m-2}{\epsilon}\epsilon}{m\alpha} = \frac{1-\epsilon}{\alpha} + \frac{m-2}{m\alpha}$. Hence, the finishing time of task τ_2 is not earlier than $\frac{1-\epsilon}{\alpha} + \frac{m-2}{m\alpha} + \frac{1-\frac{1}{\alpha}}{\alpha}$.

If $\frac{1-\epsilon}{\alpha} + \frac{m-2}{m\alpha} + \frac{1-\frac{1}{\alpha}}{\alpha} > 1 + \frac{\epsilon}{\alpha}$, then task τ_2 misses its deadline. By Lemma 9, we reach the conclusion. □

Figure 5 illustrates the upper bound of G-EDF provided in Theorem 4 and the lower bound in Theorem 5 with respect to the capacity augmentation bound. It can be easily seen that the upper and lower bounds are getting closer when m is larger. When m is 100, the gap between the upper and the lower bounds is roughly about 0.00452.

It is important to note that the more precise speedup in Corollary 2 is tight even for small m. This is because in the above example task set, $U_{\sum} = m$, total high task utilization $U_{\mathrm{H}} = m-1$ and number of heavy task $\|\tau_{\mathrm{H}}\| = 1$, then according to the Corollary, the capacity augmentation bound for this task set under G-EDF is

$$\frac{2+\frac{U_{\sum}-\|\tau_{\mathrm{H}}\|-1}{m}+\sqrt{\frac{4(U_{\mathrm{H}}-\|\tau_{\mathrm{H}}\|)}{m}+\frac{(U_{\sum}-\|\tau_{\mathrm{H}}\|-1)^2}{m^2}}}{2}$$
$$= \frac{2+\frac{m-1-1}{m}+\sqrt{\frac{4(m-1-1)}{m}+\frac{(m-1-1)^2}{m^2}}}{2} = \frac{3-\frac{2}{m}+\sqrt{5-\frac{12}{m}+\frac{4}{m^2}}}{2}$$

which is exactly the lower bound in Theorem 5 for $\epsilon \to^+ 0$.

VII. G-RM Scheduling

This section proves that G-RM provides a capacity augmentation bound of $2+\sqrt{3}$ for large m. The structure of the proof is very similar to the analysis in Section VI.

Again, we use a lemma from [15], restated below.

Lemma 10. *If* $\forall t > 0$, $0.5(\alpha \cdot m - m + 1)t \geq \sum_i work_i(t, \alpha)$ *the task set is schedulable by G-RM on speed-*α *cores.*

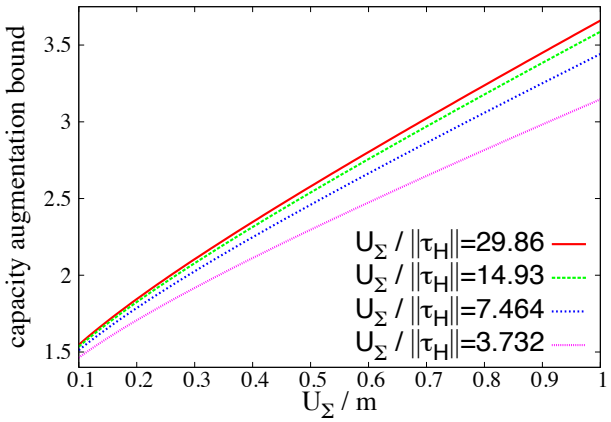


Fig. 6: The required speedup of G-RM when m is sufficiently large and $U_{\mathrm{L}} = 0$ (i.e. $U_{\sum} = U_{\mathrm{H}}$).

Proof: This is based on a reformulation of Lemma 6 and Definition 10 in [15]. Note that the analysis in [15] is for deadline-monotonic scheduling, by giving a sub-job of a task higher priority if its relative deadline is shorter. As we consider tasks with implicit deadlines, deadline-monotonic scheduling is the same as rate-monotonic scheduling. □

Proofs like those in Section VI-A give us the bounds below for G-RM scheduling.

Theorem 6. *The capacity augmentation bound for G-RM is* $\frac{4-\frac{3}{m}+\sqrt{12-\frac{20}{m}+\frac{9}{m^2}}}{2}$ *($\approx 2+\sqrt{3}$, when m is large).*

Proof: (Similar to the proof of Theorem 4:) First, we know from Lemma 7 that $\forall t > 0$, $\sum_{i=1}^{n} work_i(t,\alpha) \leq (\frac{\alpha m-\alpha}{\alpha-1})t$; Second, if $\frac{\alpha\cdot m-\alpha}{\alpha-1} \leq 0.5(\alpha m - m + 1)$. we can also conclude that the schedulability test for G-RM in Lemma 10 holds. By solving the inequality above, we have $\alpha \geq \frac{4-\frac{3}{m}+\sqrt{12-\frac{20}{m}+\frac{9}{m^2}}}{2}$, and prove Theorem 6. □

The result in Theorem 6 is the best known result for the capacity augmentation bound for global fixed-priority scheduling for general DAG tasks with arbitrary structures. Interestingly, Kim et al. [31] get the same bound of $2+\sqrt{3}$ for global fixed-priority scheduling of parallel synchronous tasks (a subset of DAG tasks).

The strategy used in [31] is quite different. In their algorithm, the tasks undergo a stretch transformation which generates a set of sequential subtask (each with its release time and deadline) for each parallel task in the original task set. These subtasks are then scheduled using a G-DM scheduling algorithm [11]. Note that even though the parallel tasks in the original task set have implicit deadlines, the transformed sequential tasks have only constrained deadlines — hence the need for deadline monotonic scheduling instead of rate monotonic scheduling.

Corollary 3. *If a task set has total utilization $U_{\sum}$, the total high task utilization U_H and the number of heavy task $\|\tau_H\|$, then this task set will be schedulable under G-RM on a m-core machine with speed* $\alpha = \frac{2+\frac{2U_{\sum}-2\|\tau_H\|-1}{m}+\sqrt{\frac{8(U_H-\|\tau_H\|)}{m}+\frac{(2U_{\sum}-2\|\tau_H\|-1)^2}{m^2}}}{2}$.

Proof: The proof is the same as in the proof of Corollary 2, except that instead of using Lemma 8, it uses Lemma 10. □

Figure 6 illustrates the required speedup of G-RM provided in Corollary 3 when m is sufficiently large (i.e., $m = \infty$ and $1/m = 0$) and $U_{\mathrm{L}} = 0$ (i.e. $U_{\sum} = U_{\mathrm{H}}$). We vary $\frac{U_{\sum}}{m}$ and $\frac{U_{\sum}}{\|\tau_{\mathrm{H}}\|}$. Again, $\frac{U_{\sum}}{\|\tau_{\mathrm{H}}\|}$ is the average utilization of all heavy tasks. It can be also be seen that the bound is getting closer to $2+\sqrt{3}$, when $\frac{U_{\sum}}{\|\tau_{\mathrm{H}}\|}$ is larger, which results from $\|\tau_{\mathrm{H}}\| = 1$ and $m = \infty$.

VIII. Practical Considerations

As have shown in the previous sections, the capacity augmentation bound for federated scheduling, G-EDF and G-RM are 2, 2.618 and 3.732, respectively. In this section, we consider their run-time efficiency and efficacy from a practical perspective. We consider four dimensions — static vs. dynamic priorities, and global vs. partitioned scheduling, overheads due to scheduling and synchronization overheads, and work-conserving vs. not.

Practitioners have generally found it easier to implement fixed (task) priority schedulers (such as RM) than dynamic priority schedulers (such as EDF). Fixed priority schedulers have always been well-supported in almost all real-time operating systems. Recently, there has been efforts on efficient implementations of job-level dynamic priority (EDF and G-EDF) schedulers for sequential tasks [16, 34]. Federated scheduling does not require any priority assignment for high-utilization tasks (since they own their cores exclusively) and can use either fixed or dynamic priority for low-utilization tasks. Thus, it is relatively easier to implement G-RM and federated scheduling.

For sequential tasks, in general, global scheduling may incur more overhead due to thread migration and the associated cache penalty, the extent of which depends on the cache architecture and the task sets. In particular, for parallel tasks, the overheads for global scheduling could be worse. For sequential tasks, preemptions and migrations only occur when a new job with higher priority is released. In contrast, for parallel tasks, a preemption and possibly migration could occur whenever a node in the DAG of a job with higher priority is enabled. Since nodes in a DAG often represent a fine-grained units of computation, the number of nodes in the task set can be larger than the number of tasks. Hence, we can expect a larger number of such events. Since federated scheduling is a generalization of partitioned scheduling to parallel tasks, it has advantages similar to partitioning. In fact, if we use a partitioned RM or partitioned EDF strategy for low-utilization tasks,

there are only preemptions but no migration for low-utilization tasks. Meanwhile, federated scheduling only allocates the minimum number of dedicated cores to ensure the schedulability of each high-utilization task, so there is no preemptions for high-utilization tasks and the number of migrations is minimized. Hence, we expect that federated scheduling will have less overhead than global schedulers.

In addition, parallel runtime systems have additional parallel overheads, such as synchronization and scheduling overheads. These overheads (per task) usually are approximately linear in the number of cores allocated to each task. Under federated scheduling, a minimum number of cores is assigned. However, depending on the particular implementation, global scheduling may execute a task on all the cores in the system and may have higher overheads.

Finally, note that federated scheduling is not a greedy (work conserving) strategy for the entire task set, although it uses a greedy schedule for each individual task. In many real systems, the worst-case execution times are normally over-estimated. Under federated scheduling cores allocated to tasks with overestimated execution times may idle due to resource over-provisioning. In contrast, work-conserving strategies (such as G-EDF and G-RM) and can utilize available cores through thread migration dynamically.

IX. Related Work

In this section, we review closely related work on real-time scheduling, concentrating primarily on parallel tasks.

Real-time multiprocessor scheduling considers scheduling sequential tasks on computers with multiple processors or cores and has been studied extensively (see [10, 23] for a survey). In addition, platforms such as Litmus^{RT} [17, 19] have been designed to support these task sets. Here, we review a few relevant theoretical results. Researchers have proven resource augmentation bounds, utilization bounds and capacity augmentation bounds. The best known resource bound for G-EDF for sequential tasks on a multiprocessor is 2 [7]; a capacity augmentation bound of $2 - \frac{1}{m} + \epsilon$ for small ϵ [14]. Partitioned EDF and versions partitioned static priority schedulers also provide a utilization bound of 2 [3, 37]. G-RM provides a capacity augmentation bound of 3 [2] to implicit deadline tasks.

For parallel real-time tasks, most early work considered intra-task parallelism of limited task models such as *malleable tasks* [22, 30, 33] and *moldable tasks* [39]. Kato et al. [30] studied the Gang EDF scheduling of moldable parallel task systems.

Researchers have since considered more realistic task models that represent programs generated by commonly used parallel programming languages such as Cilk family [13, 21], OpenMP [41], and Intel's Thread Building Blocks [43]. These languages and libraries support primitives such as parallel for-loops and fork/join or spawn/sync in order to expose parallelism within the programs. Using these constructs generates tasks whose structure can be represented with different types of DAGs.

Tasks with *parallel synchronous tasks* have been studied more than others in the real-time community. These tasks are generated if we use only parallel-for loops to generate parallelism. Lakshmanan et al. [32] proved a (capacity) augmentation bound of 3.42 for a restricted synchronous task model which is generated when we restrict each parallel-for loop in a task to have the same number of iterations. General synchronous tasks (with no restriction on the number of iterations in the parallel-for loops), have also been studied [4, 31, 40, 45]. (More details on these results were presented in Section I) Chwa et al. [20] provide a response time analysis.

If we do not restrict the primitives used to parallel-for loops, we get a more general task model — most easily represented by a general directed acyclic graph. A resource augmentation bound of $2 - \frac{1}{m}$ for G-EDF was proved for a single DAG with arbitrary deadlines [8] and for multiple DAGs [15, 35]. A capacity augmentation bound of $4 - \frac{2}{m}$ was proved in [35] for tasks with for implicit deadlines. Liu et al. [36] provide a response time analysis for G-EDF.

There has been significant work on scheduling non-real-time parallel systems [5, 6, 24–26, 42]. In this context, the goal is generally to maximize throughput. Various provably good scheduling strategies, such as list scheduling [18, 27] and work-stealing [12] have been designed. In addition, many parallel languages and runtime systems have been built based on these results. While multiple tasks on a single platform have been considered in the context of fairness in resource allocation [1], none of this work considers real-time constraints.

X. Conclusions

In this paper, we consider parallel tasks in the DAG model and prove that for parallel tasks with implicit deadlines the capacity augmentation bounds of federated scheduling, G-EDF and G-RM are 2, 2.618 and 3.732 respectively. In addition, the bound 2 for federated scheduling and the bound of 2.618 for the G-EDF are both tight for large m, since there exist matching lower bounds. Moreover, the three bounds are the best known bounds for these schedulers for DAG tasks.

There are several directions of future work. The G-RM capacity augmentation bound is not known to be tight. The current lower bound of G-RM is 2.668, inherited from the sequential sporadic real-time tasks without DAG structures [38]. Therefore, it is worth investigating a matching lower bound or lowering the upper bound. In addition, since the lower bound of any scheduler is $2 - \frac{1}{m}$, it would be interesting to investigate if it is possible to design schedulers that reach this bound. Finally, all the known

capacity augmentation bound results are restricted to implicit deadline tasks; we would like to generalize them to constrained and arbitrary deadline tasks.

Acknowledgment

This research was supported in part by the priority program "Dependable Embedded Systems" (SPP 1500 - spp1500.itec.kit.edu), by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/), by NSF grants CCF-1136073 (CPS) and CCF-1337218 (XPS). The authors thank anonymous reviewers for their suggestions on improving this paper.

References

[1] K. Agrawal, C. E. Leiserson, Y. He, and W. J. Hsu. "Adaptive work-stealing with parallelism feedback". In: *ACM Trans. Comput. Syst.* 26 (2008), pp. 112–120.

[2] B. Andersson, S. Baruah, and J. Jonsson. "Static-priority scheduling on multiprocessors". In: *RTSS*. 2001.

[3] B. Andersson and J. Jonsson. "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%". In: *ECRTS*. 2003.

[4] B. Andersson and D. de Niz. "Analyzing Global-EDF for Multiprocessor Scheduling of Parallel Tasks". In: *Principles of Distributed Systems*. 2012, pp. 16–30.

[5] N. S. Arora, R. D. Blumofe, and C. G. Plaxton. "Thread Scheduling for Multiprogrammed Multiprocessors". In: *SPAA*. 1998.

[6] N. Bansal, K. Dhamdhere, J. Konemann, and A. Sinha. "Nonclairvoyant Scheduling for Minimizing Mean Slowdown". In: *Algorithmica* 40.4 (2004), pp. 305–318.

[7] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. "Improved Multiprocessor Global Schedulability Analysis". In: *Real-Time Syst.* 46.1 (2010), pp. 3–24.

[8] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. "A generalized parallel task model for recurrent real-time processes". In: *RTSS*. 2012.

[9] S. K. Baruah, A. K. Mok, and L. E. Rosier. "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor". In: *RTSS*. 1990.

[10] M. Bertogna and S. Baruah. "Tests for global EDF schedulability analysis". In: *Journal of System Architecture* 57.5 (2011).

[11] M. Bertogna, M. Cirinei, and G. Lipari. "New Schedulability Tests for Real-time Task Sets Scheduled by Deadline Monotonic on Multiprocessors". In: *Proceedings of the 9th International Conference on Principles of Distributed Systems*. 2006.

[12] R. D. Blumofe and C. E. Leiserson. "Scheduling multithreaded computations by work stealing". In: *Journal of the ACM* 46.5 (1999), pp. 720–748.

[13] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. "Cilk: An Efficient Multithreaded Runtime System". In: *PPoPP*. 1995, pp. 207–216.

[14] V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. "A constant-approximate feasibility test for multiprocessor real-time scheduling". In: *Algorithmica* 62.3-4 (2012), pp. 1034–1049.

[15] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. "Feasibility Analysis in the Sporadic DAG Task Model". In: *ECRTS*. 2013.

[16] B. B. Brandenburg and J. H. Anderson. "On the Implementation of Global Real-Time Schedulers". In: *RTSS*. 2009.

[17] B. B. Brandenburg, A. D. Block, J. M. Calandrino, U. Devi, H. Leontyev, and J. H. Anderson. *LITMUS RT: A Status Report*. 2007.

[18] R. P. Brent. "The Parallel Evaluation of General Arithmetic Expressions". In: *Journal of the ACM* (1974), pp. 201–206.

[19] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson. "LITMUS^{RT}: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers". In: *RTSS*. 2006.

[20] H. S. Chwa, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin. "Global EDF Schedulability Analysis for Synchronous Parallel Tasks on Multicore Platforms". In: *ECRTS*. 2013.

[21] *CilkPlus*. http://software.intel.com/en-us/articles/intel-cilk-plus.

[22] S. Collette, L. Cucu, and J. Goossens. "Integrating job parallelism in real-time scheduling theory". In: *Information Processing Letters* 106.5 (2008), pp. 180–187.

[23] R. I. Davis and A. Burns. "A survey of hard real-time scheduling for multiprocessor systems". In: *ACM Computing Surveys* 43 (2011), 35:1–44.

[24] X. Deng, N. Gu, T. Brecht, and K. Lu. "Preemptive Scheduling of Parallel Jobs on Multiprocessors". In: *SODA*. 1996.

[25] M. Drozdowski. "Real-time scheduling of linear speedup parallel tasks". In: *Inf. Process. Lett.* 57 (1996), pp. 35–40.

[26] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. "Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics". In: *Journal of Scheduling* 6.3 (2003), pp. 231–250.

[27] R. L. Graham. "Bounds on Multiprocessing Anomalies". In: *SIAM Journal on Applied Mathematics* (1969), 17(2):416–429.

[28] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke. "Cyber-physical systems for real-time hybrid structural testing: a case study". In: *International Conference on Cyber Physical Systems*. 2010.

[29] A. Ka and L. Mok. *Fundamental design problems of distributed systems for the hard-real-time environment*. Tech. rep. 3. 1983.

[30] S. Kato and Y. Ishikawa. "Gang EDF Scheduling of Parallel Task Systems". In: *RTSS*. 2009.

[31] J. Kim, H. Kim, K. Lakshmanan, and R. Rajkumar. "Parallel scheduling for cyber-physical systems: analysis and case study on a self-driving car". In: *ICCPS*. 2013.

[32] K. Lakshmanan, S. Kato, and R. R. Rajkumar. "Scheduling Parallel Real-Time Tasks on Multi-core Processors". In: *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*. RTSS. 2010.

[33] W. Y. Lee and H. Lee. "Optimal Scheduling for Real-Time Parallel Tasks". In: *IEICE Transactions on Information Systems* E89-D.6 (2006), pp. 1962–1966.

[34] J. Lelli, G. Lipari, D. Faggioli, and T. Cucinotta. "An efficient and scalable implementation of global EDF in Linux". In: *OSPERT*. 2011.

[35] J. Li, K. Agrawal, C.Lu, and C. Gill. "Analysis of Global EDF for Parallel Tasks". In: *ECRTS*. 2013.

[36] C. Liu and J. Anderson. "Supporting Soft Real-Time Parallel Applications on Multicore Processors". In: *RTCSA*. 2012.

[37] J. M. López, J. L. Díaz, and D. F. García. "Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems". In: *Real-Time Systems* 28.1 (2004), pp. 39–68.

[38] L. Lundberg. "Analyzing Fixed-Priority Global Multiprocessor Scheduling". In: *IEEE Real Time Technology and Applications Symposium*. 2002, pp. 145–153.

[39] G. Manimaran, C. S. R. Murthy, and K. Ramamritham. "A New Approach for Scheduling of Parallelizable Tasks inReal-Time Multiprocessor Systems". In: *Real-Time Syst.* 15 (1998), pp. 39–60.

[40] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. "Techniques optimizing the number of processors to schedule multi-threaded tasks". In: *ECRTS*. 2012.

[41] *OpenMP Application Program Interface v3.1*. http://www.openmp.org/mp-documents/OpenMP3.1.pdf. 2011.

[42] C. D. Polychronopoulos and D. J. Kuck. "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers". In: *Computers, IEEE Transactions on* C-36.12 (1987).

[43] J. Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, 2010.

[44] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. Gill. "Parallel real-time scheduling of DAGs". In: *IEEE Transactions on Parallel and Distributed Systems* (2014).

[45] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill. "Multi-core real-time scheduling for generalized parallel task models". In: *Real-Time Systems* 49.4 (2013), pp. 404–435.

[46] R. Sedgewick and K. D. Wayne. *Algorithms*. 4th. 2011.